|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт информационных технологий |
| Кафедра вычислительной техники |
|  |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ** | |
| **по дисциплине** | |
| **«Структуры и алгоритмы обработки данных»** | |
|  | |
| Выполнил студент группы ИКБО-03-17 | Сагдеев Р.Р. |
| Принял | Асадова Ю.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторные работы выполнены | «\_\_»\_\_\_\_\_\_\_2019 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2019 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя) |

Москва 2019

|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт информационных технологий

Кафедра вычислительной техники

Выполнено \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Сагдеев Р.Р./

Зачтено \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Асадова Ю.С./

**Задание на лабораторную работу №1**

Дисциплина: **«Структуры и алгоритмы обработки данных»**

Студент Сагдеев Руста Робертович Группа ИКБО-03-17

**1. Тема**: «Ветвящиеся вычислительные процессы».

**2. Срок сдачи студентом законченной работы:** 28 февраля 2019 г.

**3. Исходные данные:** произвольное число.

**4. Задание:** .

|  |  |  |
| --- | --- | --- |
| 17 | Занести цифры числа в стек и вывести на экран число, цифры которого идут в обратном порядке. |  |

**5. Содержание отчета:**

* титульный лист;
* задание;
* оглавление;
* введение;
* основные разделы отчета;
* заключение;
* список использованных источников;

Руководитель работы Асадова Ю.С. \_\_\_\_\_\_\_\_\_\_\_\_ «14» февраля 2019г.

подпись

Задание принял к исполнению Сагдеев Р.Р. \_\_\_\_\_\_\_\_\_\_\_\_ «14» февраля 2019г.

подпись

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc2263846)

[**ОСНОВНЫЕ РАЗДЕЛЫ ОТЧЁТА** 6](#_Toc2263847)

[**ПРИМЕР РАБОТЫ ПРОГРАММЫ** 8](#_Toc2263848)

[**ЗАКЛЮЧЕНИЕ** 10](#_Toc2263849)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 11](#_Toc2263850)

[**ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ №2** 1](#_Toc5574862)

[**ВВЕДЕНИЕ** 3](#_Toc5574863)

[**1.РАЗБОР ПРОГРАММЫ** 4](#_Toc5574864)

[**2.ИСХОДНЫЙ КОД ПРОГРАММЫ** 4](#_Toc5574864)

[**3.ПРИМЕР РАБОТЫ ПРОГРАММЫ** 4](#_Toc5574864)

[**ЗАКЛЮЧЕНИЕ** 12](#_Toc5574865)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 13](#_Toc5574866)

# **ВВЕДЕНИЕ**

Связный список является простейшим типом данных динамической структуры, состоящей из элементов (узлов). Каждый узел включает в себя в классическом варианте два поля: данные (в качестве данных может выступать переменная, объект класса или структуры и т. д.) и указатель на следующий узел в списке.

Элементы связанного списка можно помещать и исключать произвольным образом.  
Доступ к списку осуществляется через указатель, который содержит адрес первого элемента списка, называемый корнем списка (head).

Классификация списков

По количеству полей указателей различают однонаправленный (односвязный) и двунаправленный (двусвязный) списки.

Связный список, содержащий только один указатель на следующий элемент, называется односвязным.

Связный список, содержащий два поля указателя – на следующий элемент и на предыдущий, называется двусвязным.

По способу связи элементов различают линейные и циклические списки.

Линейные и циклические списки

Связный список, в котором, последний элемент указывает на NULL, называется линейным.

Связный список, в котором последний элемент связан с первым, называется циклическим.

Таким образом, различают 4 основных вида списков.

Односвязный линейный список (ОЛС).

Каждый узел ОЛС содержит 1 поле указателя на следующий узел. Поле указателя последнего узла содержит нулевое значение (указывает на NULL).

Односвязный циклический список (ОЦС).

Каждый узел ОЦС содержит 1 поле указателя на следующий узел. Поле указателя последнего узла содержит адрес первого узла (корня списка).

Двусвязный линейный список (ДЛС).

Каждый узел ДЛС содержит два поля указателей: на следующий и на предыдущий узел. Поле указателя на следующий узел последнего узла содержит нулевое значение (указывает на NULL). Поле указателя на предыдущий узел первого узла (корня списка) также содержит нулевое значение (указывает на NULL).

Двусвязный циклический список (ДЦС).

Каждый узел ДЦС содержит два поля указателей: на следующий и на предыдущий узел. Поле указателя на следующий узел последнего узла содержит адрес первого узла (корня списка). Поле указателя на предыдущий узел первого узла (корня списка) содержит адрес последнего узла.

1. **Разбор программы**

Данная программа заносит цифры введенного числа в стек, а затем выводит число, у которого цифры идут в обратном порядке. На выходе программа выдает новое число с обратным порядком цифр и выводит его на экран.

1. **Исходный код программы**

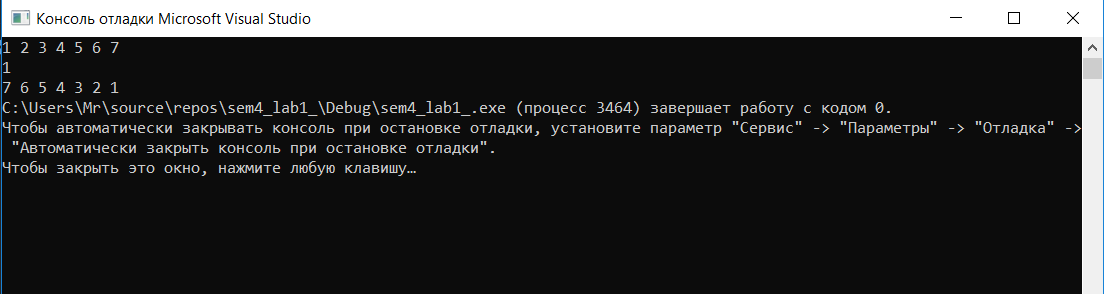
Main.cpp:

|  |
| --- |
| #include "pch.h"  #include <iostream>  using namespace std;  class Stack  {  private:  static const int MAX = 7;  int st[MAX];  int top;  public:  Stack() : top(0)  {  for (int i = 0; i < MAX; ++i)  st[i] = 0;  }  void push(int var)  {  if (top < MAX)  st[top++] = var;  else  std::cerr << " error: Stack is full! ";  }  int pop()  {  if (top)  return st[--top];  else {  std::cerr << " error: Stack is empty! ";  return 0;  }  }  bool isFull() const { return top == MAX; }  size\_t size() const { return top; }  friend istream& operator >>(istream& is, Stack& s)  {  int k = 0;  while (is >> k && !(s.isFull())) {  s.push(k);  }  return is;  }  friend ostream& operator <<(ostream& os, const Stack& s)  {  int k = s.top;  if (k)  for (int i = k - 1; i >= 0; --i)  os << s.st[i] << ' ';  else  os << "error: Stack is empty!";  return os;  }  };  int main()  {  Stack s;  cin >> s;  cout << s;  return 0;  } |

|  |
| --- |
|  |

# **Пример работы программы**

На рисунке 1 показан пример работы программы с корректными введенными данными. Программа принимает на вход число подходящего размера, после чего заносит в стек все цифры числа и выводит на экран новое число, с цифрами в обратном порядке.



*Рисунок 1 – Пример работы программы*

# **ЗАКЛЮЧЕНИЕ**

В данной лабораторной работе мы научились работать с одномерными списками в c++, вводить/выводить данные из списков, создавать их и работать с ними. Изучили синтаксис их объявления и представление их в памяти компьютера.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Павловская Т.А., Щупак Ю.А. С/С++. Структурное программирование. Практикум. – СПб.: Питер, 2002.
2. Подбельский В.В. Язык С++; Учебное пособие. М.: Финансы и статистика, 1996.
3. Павловская Т.А. С/С++. Программирование на языке высокого уровня. – СПб.: Питер, 2002.
4. Дейтл Х.М., Дейтл П.Дж. Как программировать на С. – М.: Бином, 2004.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт информационных технологий

Кафедра вычислительной техники

Выполнено \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Сагдеев Р.Р./

Зачтено \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Асадова Ю.С./

**Задание на лабораторную работу №2**

Дисциплина: **«Структуры и алгоритмы обработки данных»**

Студент Сагдеев Рустам Робертович Группа ИКБО-03-17

**1. Тема**: «Древовидные структуры».

**2. Срок сдачи студентом законченной работы:** 28 марта 2019 г.

**3. Исходные данные:** Вводимые значения для узлов дерева

**4. Задание:** Составить программу для построения и обработки упорядоченного двоичного дерева, содержащего узлы целого типа. Реализовать добавление нового узла, положение нового узла определяется в соответствии с требованием сохранения порядка.

**5. Содержание отчета:**

* титульный лист;
* задание;
* оглавление;
* введение;
* основные разделы отчета;
* заключение;
* список использованных источников;

Руководитель работы Асадова Ю.С. \_\_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2019г.

подпись

Задание принял к исполнению Сагдеев Р.Р. \_\_\_\_\_\_\_\_\_\_\_\_ «14» марта 2019г.

подпись

# **ВВЕДЕНИЕ**

В данной лабораторной работе проверяются умения студентов реализации бинарного дерева, типовых операций с бинарными деревьями, а также использования их при решении задач, требующих оптимизации ресурсов времени и памяти.

Древовидные структуры данных представляют собой динамические связные структуры, отличающиеся от списков тем, что система связей не носит линейного характер, а образует ветви, подобно природному дереву. Эти структуры можно разделить на две группы – деревья и пирамиды.

Дерево – структура данных, представляющая собой древовидную структуру в виде набора связанных узлов.

**Элементы деревьев**

Как форма организации данных, дерево является связанной структурой, где с каждым элементом данных, кроме одного, связан один узел-родитель и может быть связано произвольное число узлов-потомков.

Если узел не имеет потомков, он называется концевым узлом, или листом. Если узел не имеет родителя, он называется корневым узлом (root).

Высота деревьев – это количество уровней деревьев. Количество ветвей, растущих из узла дерева, называется степенью исхода узла.

**Классификация деревьев**

Деревья делятся на две группы – двоичные (бинарные) и с числом ветвей больше 2 – мульти вариантные. В общем случае данные, хранящиеся в дереве, не обязаны быть упорядоченными каким-либо образом.

Основные отличия – у списка соседними элементами являются предшествующий и последующий элементы, и структура линейна; а у двоичного дерева поиска соседними элементами являются с меньшим и большим ключом, и структура в общем случае ветвящиеся - в виде дерева. Также любая часть дерева по своей структуре повторяет все дерево в целом, т.е. дерево – это рекурсивная структура данных.

**Представление деревьев в памяти**

Считаем, что деревом называется сетевая структура, обладающая следующими свойствами:

* Все нелистовые вершины содержат одно поле и 2 поддерева или 2 поля и 3 поддерева.
* Все листовые вершины находятся на одном уровне (на нижнем уровне) и содержат 1 или 2 поля.
* Все данные отсортированы (по принципу двоичного дерева поиска).
* Нелистовые вершины содержат одно или два поля, указывающие на диапазон значений в их поддеревьях. Значение первого поля строго больше наибольшего значения в левом поддереве и меньше или равно наименьшему значению в правом поддереве (или в центральном поддереве, если это 3-вершина); аналогично, значение второго поля (если оно есть) строго больше наибольшего значения в центральном поддереве и меньше или равно, чем наименьшее значение в правом поддереве. Эти нелистовые вершины используются для направления функции поиска к нужному поддереву и, в конечном итоге, к нужному листу.

1. **РАЗБОР ПРОГРАММЫ**

При запуске программы пользователь должен ввести значения узлов дерева, тем самым создав его.

Далее программа выводит получившееся бинарное дерево. Затем программа просит пользователя ввести значение добавляемого узла в дерево.

На выходе программа выводит новое бинарное дерево.

1. **ИСХОДНЫЙ КОД ПРОГРАММЫ**

**#include "stdafx.h"**

**#include <iostream>**

**#include <cmath>**

**#include <algorithm>**

**struct Tree**

**{**

**int x;**

**Tree \*l, \*r;**

**};**

**void show(Tree \*&s, int n)**

**{**

**n++;**

**if (s != NULL)**

**{**

**for (int i = 0; i < n; i++)**

**std::cout << " ";**

**std::cout << s->x << std::endl;**

**show(s->l, n);**

**show(s->r, n);**

**}**

**show(f, -1);**

**mirror(f);**

**show(f, -1);**

**std::cout << maxDepth(f) << std::endl;**

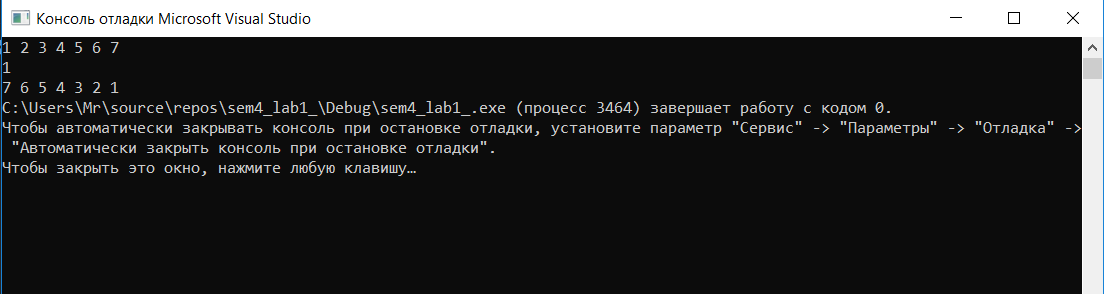
**system("pause");**

**return 0;**

**}**

|  |
| --- |
|  |

# **Пример работы программы**

На рисунке 1 показывается ввод результата работы программы. Пользователь вводит значения узлов дерева, после добавляет новый узел, вводя его значение 

*Рисунок 1 – Пример работы программы*

|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт информационных технологий

Кафедра вычислительной техники

Выполнено \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Сагдеев Р.Р./

Зачтено \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Асадова Ю.С./

**Задание на лабораторную работу №3**

Дисциплина: **«Структуры и алгоритмы обработки данных»**

Студент Сагдеев Рустам Робертович Группа ИКБО-03-17

**1. Тема**: «Древовидные структуры».

**2. Срок сдачи студентом законченной работы:** 06 апреля 2019 г.

**3. Исходные данные:** Вводимые значения для узлов дерева

**4. Задание:** Составить программу которая определяет N периферию для данного N, при условии что дана двусторонних дорог N- периферии называется множество городов, расстояние от которых до выделенного города (столицы) больше N.

**5. Содержание отчета:**

* титульный лист;
* задание;
* оглавление;
* введение;
* основные разделы отчета;
* заключение;
* список использованных источников;

Руководитель работы Асадова Ю.С. \_\_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2019г.

подпись

Задание принял к исполнению Сагдеев Р.Р. \_\_\_\_\_\_\_\_\_\_\_\_ «06» апреля 2019г.

подпись

# **ВВЕДЕНИЕ**

В данной лабораторной работе проверяются умения студентов реализации типовых операций со структурами данных, реализации типовых операций с ними, выработка навыков применения графовых структур в задачах.

Графы обычно применяются для моделирования или описания систем или сообщений – топографических карт, маршрутов транспорта или потоков грузов. Сообщений и т.п. технологических процессов, отношений между объектами. При этом обычно требуется описывать вершины графа как элементы , хранящие некоторую информацию.

**Алгоритмы на графах**

Графы предоставляют механизм формализации многих задач, в которых в основе постановки лежит набор отношений объектов, например задач по оптимизации систем задач дорог, сетей взаимодействующих объектов, задач имеющих дело с программными или лингвистическими моделями. В наиболее распространенных случаях формулирование задачи в терминах теории графов .

Наряду с операциями характерными и для списков и деревьев – добавлением и удалением вершин – для графов существуют аналогичные операции для ребер поиск вершин, обход графа. Имеются специфические для графов операции: построение остовного дерева, поиск пути.

1. **РАЗБОР ПРОГРАММЫ**

Программа получает на вход количество вершин в графе. Затем программа автоматически формирует граф с указанным количеством вершин в виде матрицы связей. И в конце выводятся номера вершин (городов), которые входят в N-периферию. Также есть следующие функции:

1. size\_t FindNearestCity() – найти номер ближайшего к выделенному городу (столице) города.
2. void FindMinPaths() – создать массив кратчайших расстояний от городов до столицы с помощью алгоритма Дейктсры.
3. std::vector<size\_t>\* FindPerifery(size\_t N, CityRoadSystem& cityRoadSystem, size\_t capitalNumber) – найти N-периферию.

Программа предлагает ввести количество вершин в графе. Затем программа генерирует значение связей между городами (от 0 до 100) и выводит симметричную относительно главной диагонали матрицу связей. После этого необходимо ввести номер выделенного города (столицы), а затем – число N. Программа отображает номера вершин (городов), которые входят в N-периферию.

1. **ИСХОДНЫЙ КОД ПРОГРАММЫ**

**#include <iostream>**

**#include <vector>**

**#include <random>**

**#include "TextTable.h"**

**//Город**

**class CityRoadSystem**

**{**

**private:**

**//Матрица связей**

**std::vector<std::vector<size\_t>> Connections;**

**public:**

**//Создать матрицу связей и заполнить её случайными вершинами**

**CityRoadSystem(size\_t citiesAmount)**

**{**

**Connections.resize(citiesAmount);**

**for (size\_t i = 0; i < citiesAmount; i++)**

**Connections[i].resize(citiesAmount);**

**for (size\_t i = 0; i < citiesAmount; i++)**

**Connections[i][i] = 0;**

**for (size\_t i = 0; i < citiesAmount; i++)**

**for (size\_t j = 0; j < i; j++)**

**Connections[i][j] = rand() % 100;**

**for (size\_t i = 0; i < citiesAmount; i++)**

**for (size\_t j = i + 1; j < citiesAmount; j++)**

**Connections[i][j] = Connections[j][i];**

**}**

**//Получить количество вершин**

**size\_t GetSize()**

**{**

**return Connections.size();**

**}**

**private:**

**CityRoadSystem\* cityRoadSystem = nullptr;**

**size\_t capitalNumber;**

**bool\* IsChecked;**

**size\_t\* Distances;**

**DeikstryNode() = delete;**

**public:**

**DeikstryNode(CityRoadSystem& cityRoadSystem, size\_t capitalNumber)**

**{**

**this->cityRoadSystem = &cityRoadSystem;**

**this->capitalNumber = capitalNumber;**

**IsChecked = new bool[cityRoadSystem.Connections.size()];**

**Distances = new size\_t[cityRoadSystem.Connections.size()];**

**for (size\_t i = 0; i < cityRoadSystem.Connections.size(); i++)**

**{**

**IsChecked[i] = false;**

**Distances[i] = UINT32\_MAX;**

**}**

**Distances[capitalNumber - 1] = 0;**

**FindMinPaths();**

**}**

**{**

**srand(time(nullptr));**

**size\_t nodesAmount;**

**size\_t N;**

**std::cout << "Enter amount of nodes: ";**

**std::cin >> nodesAmount;**

**CityRoadSystem cityRoadSystem(nodesAmount);**

**cityRoadSystem.ShowGraph();**

**size\_t capitalNumber;**

**std::cout << "Enter the number of capital (from 1): ";**

**std::cin >> capitalNumber;**

**std::cout << "Enter N: ";**

**std::cin >> N;**

**std::vector<size\_t>\* foundCities = FindPerifery(N, cityRoadSystem, capitalNumber);**

**if (foundCities->empty())**

**std::cout << "Cities are not found" << std::endl;**

**else**

**{**

**std::cout << "N-perifery: ";**

**for (size\_t i = 0; i < foundCities->size(); i++)**

**std::cout << (\*foundCities)[i] << " ";**

**std::cout << std::endl;**

**}**

**delete foundCities;**

**system("PAUSE");**

**return 0;**

**}**

1. **Пример работы программы**

В первом примере программа отображает номера вершин, которые входят в N-периферию.

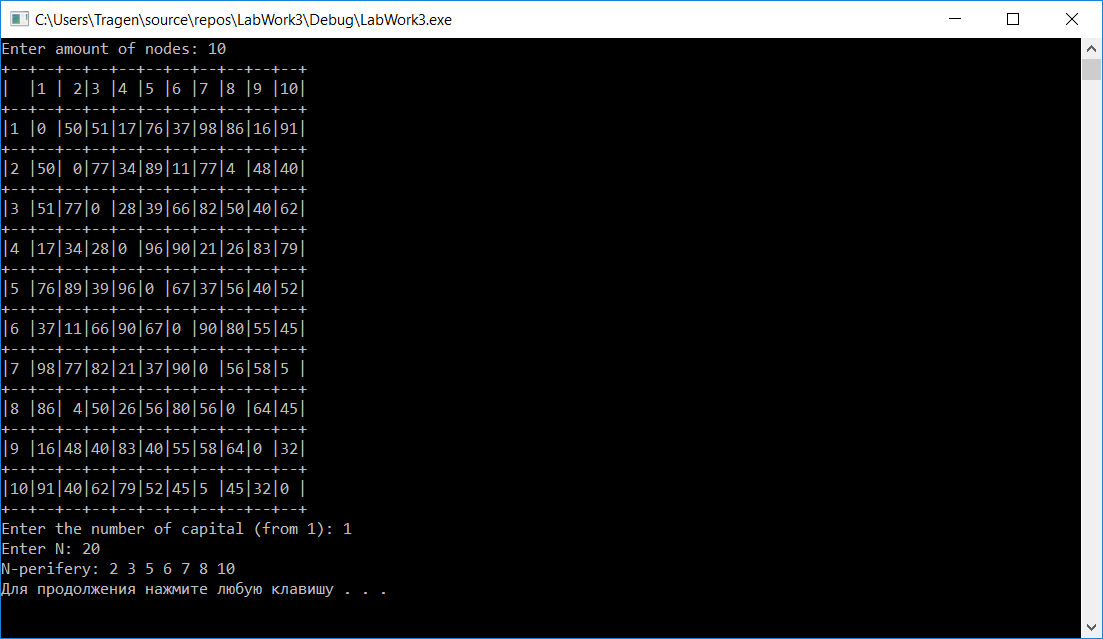


Рисунок 1 – Первый тест программы.

Во втором примере, программа выводи результат об отсутствии вершин, входящих в N-периферию.

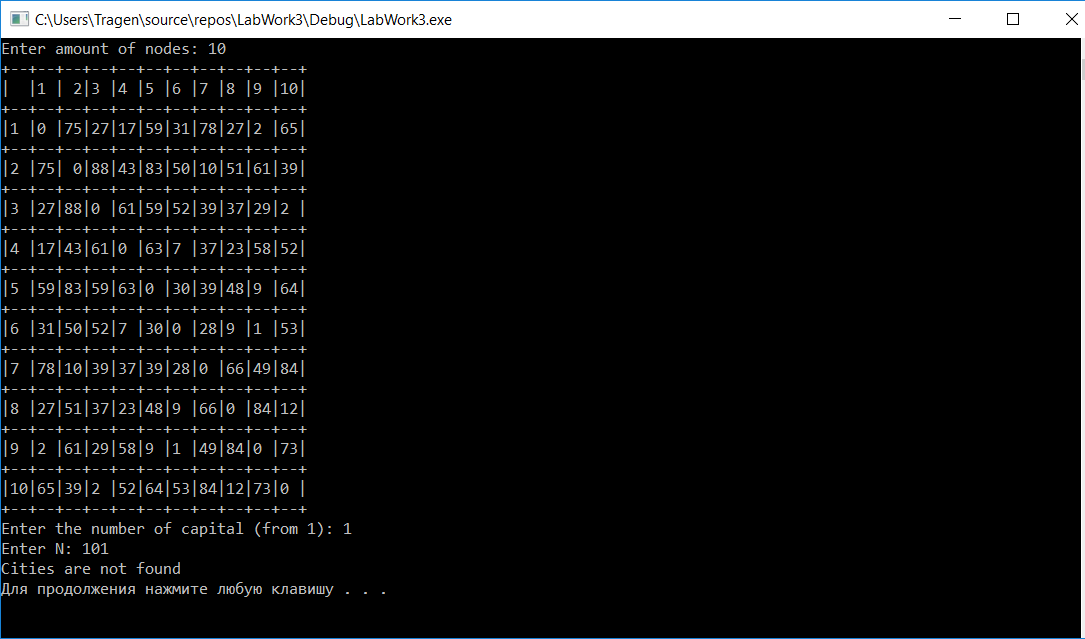


Рисунок 2 – Второй результат программы.